A Bayesian Approach to Predicting NFL Quarterback Scores in Fanduel Tournaments

STAT 578, Fall 2017, ***Team 5****: Aaron Ray, Kiomars Nassiri, Michael Chan*

October 25, 2017

Project Description

The National Football League (NFL), being one of the major professional sports leagues in North America, has a wide audience. participates in the NFL craze by competing in fantasy football tournaments organized by the daily fantasy site, "FanDuel.com". Participants in a **Fantasy Football** game act as the managers of a virtual footbal team and try to maximize their points by picking up the best line-up. Points are given based on actual performance of players in real-world competition. For the purpose of this project we have chosen to work with the data gathered from the **FanDuel** internet company. We will leverage a Hierarchical Bayesian approach with the Markov Chain Monte Carlo method to predict the fantasy points likely to be scored by an NFL quarterback in any given game. The goal is to predict the points scored by each player given certain prior conditions and predictor variables that will assist our model in providing credible posterior prediction intervals.

The analysis is inspired by the study presented in the article, **Bayesian Hierarchical Modeling Applied to Fantasy Football Projections for Increased Insight and Confidence**, by Scott Rome.

Team Members

- **Aaron Ray** (aaronwr2@illinois.edu)*
- **Kiomars Nassiri** (nassiri2@illinois.edu)
- **Michael Chan** (mhchan3@illinois.edu)

*Contact Person

Dataset Description

Team has set up a process to gather the historical data from the RotoGuru website. The following is the code used to get the data from RotoGuru:

```
# Scrape rotoguru1 site for weekly FanDuel stats and bind each week's data to the
# pre-defined dataframe, 'd'.

for(year in 2014:2017){
  for(week in 1:16){
    page = read_html(
      gsub(" ","",
           paste("http://rotoguru1.com/cgi-bin/fyday.pl?week=",week,"&year=",
                 year ,"&game=fd&scsv=1")
      ))
    dtext = page %>% html_nodes("pre") %>% html_text(trim = TRUE)
    dtable = read.table(text=dtext, sep = ";", header=TRUE, col.names = cnames,
```

```
                        quote=NULL)
    d = rbind(d,dtable)
  }
}
```

Data cleaning is performed using R routines. Some data cleaning tasks are needed to calculate Player rank.

Response Variables

- `FanDuelPts`: Points position at the end of a single game

Predictor Variables

- `AvgPts5Wks`: The 5 game average points of the player
- `AvgOppPAP7Wks` : The 7 game average Opposing Points Allowed to Position (OppPAP) by the current player's opposing defense. For example, if the Buffalo Bills defense allowed a total of 30 points per game to wide receivers for six games straight, then this number would equal to the average of 30 for any wide receiver facing the Bills defense.
- `Position`: The position the player plays
- `HomeGame`: Whether it is home game.
- `Rank`: The rank of a player based on recent performance

Analysis Ideas

Model

At the lowest level, we model the performance (`FanDuelPts`) as normally-distributed around a true value:

$y|\alpha, \beta_{defense}, \beta_{home}, \beta_{away}, \sigma_r^2 \sim N(\alpha + X_{defense}.\beta_{defense} + X_{home}.\beta_{home} + X_{away}.\beta_{away}, \sigma_y^2 I)$

where

α = The average fan duel point of the previous 5 weeks of the player, `AvgPts5Wks`

$\beta_{defense,p}$ = defense coefficient against team t for position p

$\beta_{home,p,r}$ = home coefficient for position p and a rank r player

$\beta_{away,p,r}$ = Away coefficient for position p and a rank r player

y = `FanDuelPts`

x_p = interaction indicator term for opposing team score allowed by position p

$x_{home,p,r}$ = interaction indicator term for rank r, position p, and whether it is home game

At higher level, we model the defense effect, $\beta_{defense}$, as how good(bad) a particular team's defense is against the player's position. We pool the effect based on the position of the player. That is, the defense coefficient is normally distributed from the same position specific distribution.

$\beta_{defense,p} \sim N(\delta_p, \sigma_\delta^2)$

where σ_δ is constant = 1000

For the home and away game effect, β_{home} and β_{away}, we model the effect for player of the same rank has the same distribution. We model the home and away game effect to be the same for players of the same position.

$\beta_{home,p,r} \sim N(\eta_r, \sigma_\eta^2)$

$\beta_{away,p,r} \sim N(\rho_r, \sigma_\rho^2)$

where σ_η, σ_ρ are constant $= 1000$

We will approximate non informative prior using:

$\sigma_y \sim Inv - gamma(0.0001, 0.0001)$

$\delta \sim N(0, 10000^2)$

$\eta \sim N(0, 10000^2)$

$\rho \sim N(0, 10000^2)$

Here is the JAGS model:

```
#sink("fdp.bug")
#cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                 + inprod(X.home[i, ], beta.home)
                 + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # The entry of the beta.defense corresponds to Opponent:Position
  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    beta.defense[p] ~ dnorm(delta[p], 1/1000^2)
    delta[p] ~ dnorm(0, 1/100000^2)
  }

  # The entry of the beta.home and beta.away corresponds to Rank:Position
  # In our model, we pool the beta.home/away based on rank
  for (r in 1:Num.Rank) {
    for (t in 1:Num.Position) {
      beta.home[(t-1) * Num.Rank + r] ~ dnorm(eta[r], 1/1000^2)
      beta.away[(t-1) * Num.Rank + r] ~ dnorm(rho[r], 1/1000^2)
    }
    eta[r] ~ dnorm(0, 1/100000^2)
    rho[r] ~ dnorm(0, 1/100000^2)
  }

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}
#    ",fill = TRUE)
#sink()
```

```
library(knitr)
```

Sample Data

```
fdp <- read.csv("fdpfinal.csv", sep = ',', header = TRUE)

head(fdp)
```

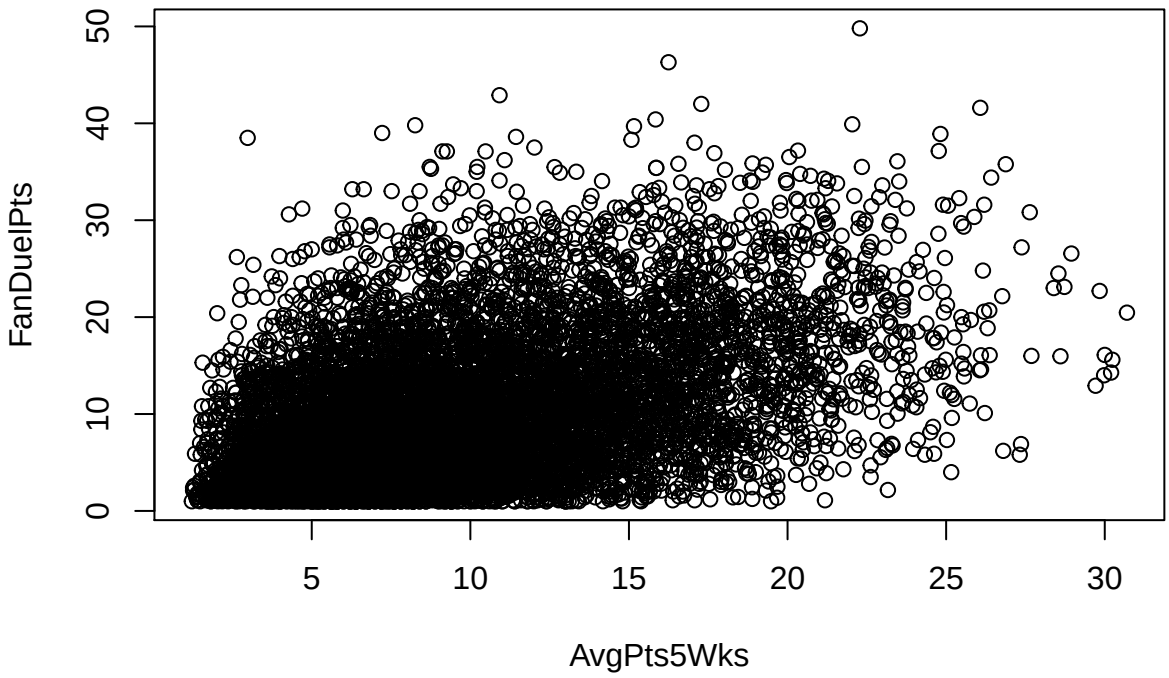
```
##   Position Year YearWeek Opponent Week PlayerId             Name    Team
## 1       QB 2015   201513 Steelers   13     1060 Hasselbeck, Matt   Colts
## 2       QB 2015   201514  Jaguars   14     1060 Hasselbeck, Matt   Colts
## 3       QB 2015   201515   Texans   15     1060 Hasselbeck, Matt   Colts
## 4       QB 2015   201516 Dolphins   16     1060 Hasselbeck, Matt   Colts
## 5       QB 2015   201501   Ravens    1     1081  Manning, Peyton Broncos
## 6       QB 2015   201502   Chiefs    2     1081  Manning, Peyton Broncos
##   HomeGame FanDuelPts FanDuelSalary AvgOppPAP7Wks SdOppPAP7Wks OallAvgPAP
## 1        0       6.86          6500         20.95        8.045      17.44
## 2        0       9.08          6600         24.16        6.738      17.44
## 3        1       8.98          6400         16.36        9.690      17.44
## 4        0       3.96          6000         20.46        6.002      17.44
## 5        1       5.90          9100         18.99       11.697      17.44
## 6        0      21.24          8200         13.85        4.342      17.44
##   OallStdevPAP AvgPts5Wks StdevPts5Wks OffRnk5Wks DefRnk7Wks
## 1        2.909      14.85        4.824      Rank4      Rank1
## 2        2.909      14.82        4.897      Rank4      Rank1
## 3        2.909      13.56        5.490      Rank4      Rank3
## 4        2.909      12.11        5.566      Rank4      Rank2
## 5        2.909      14.96        8.496      Rank3      Rank1
## 6        2.909      10.53        5.011      Rank4      Rank4
```

```
fdp['Rank'] = fdp$OffRnk5Wks
fdp['Locality'] = 'Away'
fdp[fdp$HomeGame == 1, 'Locality'] = 'Home'
```

Simple Ideas

$y|\alpha \sim N(\alpha, \sigma_y^2 I)$

```
mod.classic = lm(FanDuelPts ~ AvgPts5Wks, data = fdp)

plot(FanDuelPts ~ AvgPts5Wks, data = fdp)
```



All over the place, let's add the team defense

`X.defense` is the indicator matrix

** Set up train data **

```
#fdp_train=fdp[fdp$Year == 2015, ]
fdp_train=fdp[fdp$Year == 2016  & ((fdp$Position == "QB"& fdp$FanDuelSalary > 6500 & !is.na(fdp$FanDuel
#fdp_train=fdp[fdp$Year == 2015 & (fdp$Position == "QB" | fdp$Position == "RB") & fdp$FanDuelSalary > 6

fdp_test =fdp_train[fdp_train$YearWeek >= 201615, ]
fdp_train=fdp_train[fdp_train$YearWeek < 201615, ]
fdp_train = droplevels(fdp_train)
```

```
Use.Rank = TRUE
Num.Opponent = length(unique(fdp_train[, "Opponent"]))
Num.Position = length(unique(fdp_train[, "Position"]))
if (Use.Rank) {
  Num.Rank = length(unique(fdp_train[, "Rank"]))
  Num.HomeAwayInit = Num.Rank
  Model.File.Ext = ""
} else {
  Num.HomeAwayInit = 1
  Model.File.Ext = ".norank"
}
```

```
if (Num.Position == 1) {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks, data=fdp_train)
  if (Use.Rank) {
    X.home = model.matrix(~ 0 + Rank , data=fdp_train)
    X.away = model.matrix(~ 0 + Rank , data=fdp_train)
  } else {
    X.home = rep(1, nrow(fdp_train))
    X.away = rep(1, nrow(fdp_train))
  }
} else {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks:Position, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position, data=fdp_train)
  if (Use.Rank) {
    X.home = model.matrix(~ 0 + Rank:Position , data=fdp_train)
    X.away = model.matrix(~ 0 + Rank:Position , data=fdp_train)
  } else {
    X.home = model.matrix(~ 0 + Position , data=fdp_train)
    X.away = model.matrix(~ 0 + Position , data=fdp_train)
  }
}


X.home = X.home * fdp_train$HomeGame
X.away = X.away * (1- fdp_train$HomeGame)
X = cbind(X.defense, X.home, X.away)
```

```
library(rjags)
set.seed(20171008)
```

```
# Initialization List for the 4 chains
jags.inits=list(
  list( sigmasqinv=    0.01,  delta = rep(-100000, Num.Position),
        eta = c(100000, -100000, 100000, -100000)[1:Num.HomeAwayInit],
        rho = c(-100000, 100000, -100000, 100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 ),
  list( sigmasqinv=    0.01,  delta = rep(100000, Num.Position),
        eta = c(100000, -100000, -100000, 100000)[1:Num.HomeAwayInit],
        rho = c(-100000, 100000, 100000, -100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 1 ),
  list( sigmasqinv=0.000001,  delta = rep(-100000, Num.Position),
        eta = c(-100000, 100000, -100000, 100000)[1:Num.HomeAwayInit],
        rho = c(100000, -100000, 100000, -100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 2 ),
  list( sigmasqinv=0.000001,  delta = rep(100000, Num.Position),
        eta = c(-100000, 100000, 100000, -100000)[1:Num.HomeAwayInit],
        rho = c(100000, -100000, -100000, 100000)[1:Num.HomeAwayInit],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 3 )
)

data.jags <- list(
  y= fdp_train$FanDuelPts,
  alpha = fdp_train$AvgPts5Wks,
```

```
  X.defense = X.defense,
  X.home = X.home,
  X.away = X.away,
  Num.Position=Num.Position,
  #Num.Opponent=Num.Opponent,
  Num.Rank=Num.Rank
)
```

```
burnAndSample = function(m, N.burnin, N.iter, show.plot, mon.col) {
  update(m, N.burnin) # burn-in

  x <- coda.samples(m, mon.col, n.iter=N.iter)

  if(show.plot) {
    plot(x, smooth=FALSE)
  }

  gelman.R = gelman.diag(x, autoburnin=FALSE, multivariate = FALSE)
  print(gelman.R)

  result <- list(
    coda.sam = x,
    gelman.R.max=max(gelman.R$psrf[, 1])
  )

  return(result)
}
```

```
runModel=TRUE
runSample=TRUE

mon.col <- c("delta", "eta", "rho", "beta.defense", "beta.home", "beta.away", "sigmasq")

NSim = 30000
if (runModel) {
  m <- jags.model("fdp.bug", data.jags, inits = jags.inits, n.chains=4, n.adapt = 1000)
  save(file=paste("fdp.jags.model.init", Model.File.Ext, ".Rdata", sep=""), list="m")
} else {
  load(paste("fdp.jags.model.init", Model.File.Ext, ".Rdata", sep=""))
  m$recompile()
}
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 746
##    Unobserved stochastic nodes: 29
##    Total graph size: 18344
##
## Initializing model
```

```r
load.module("dic")
```

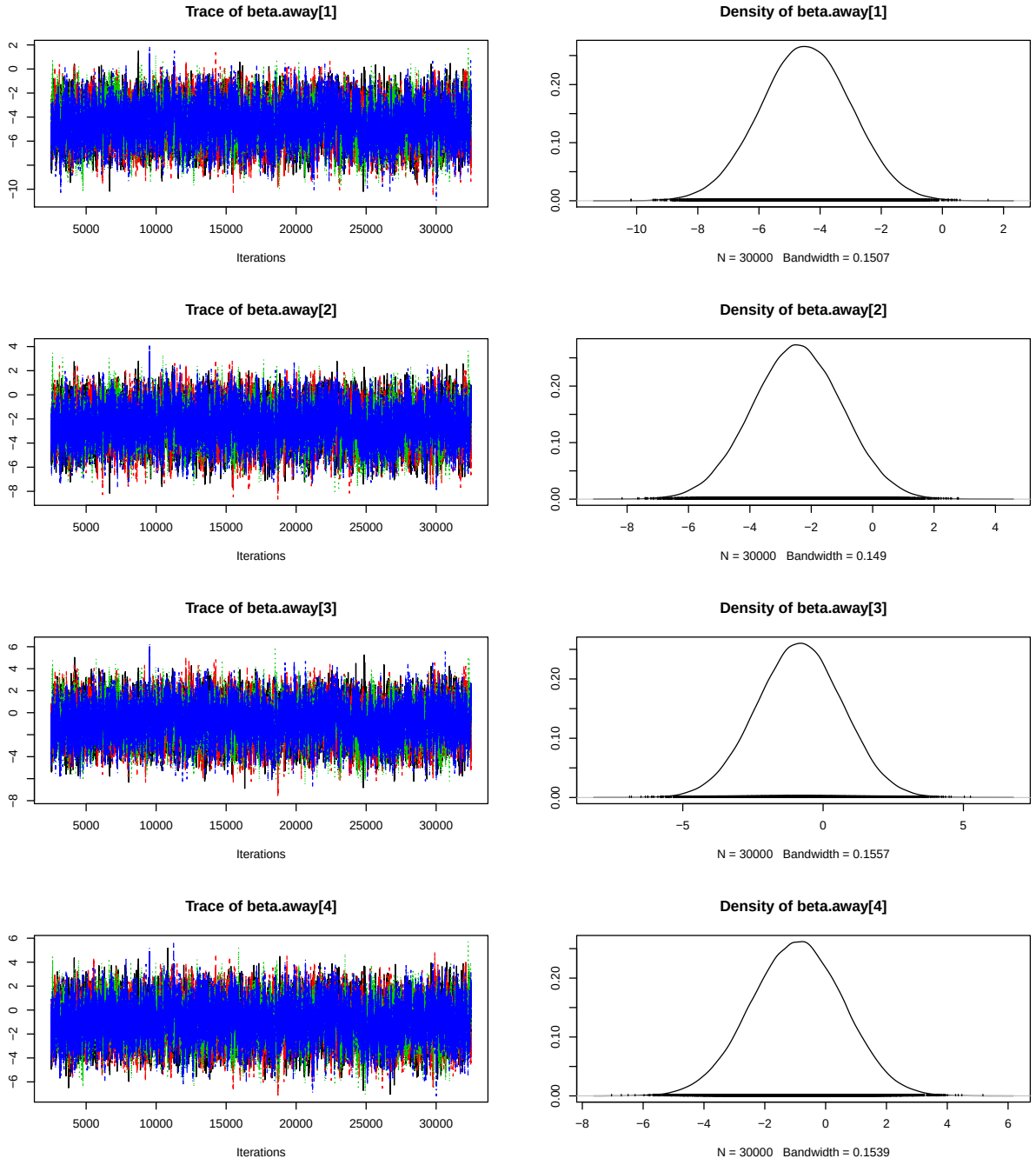
```
## module dic loaded
```

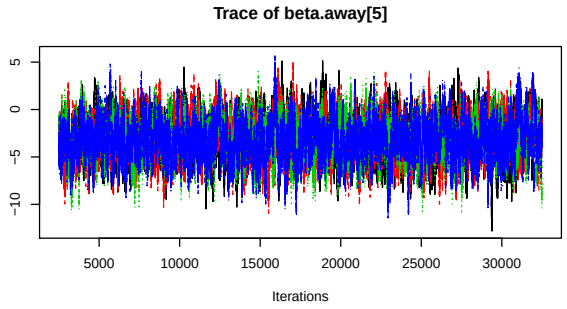
```r
N.Retry.Loop = 1
if (runSample) {
  N.burnin=2500/2
  for (loopIdx in 1:N.Retry.Loop) {
    (start_time <- Sys.time())
    (N.burnin = N.burnin * 2)
    result = burnAndSample(m, N.burnin, NSim, show.plot=FALSE, mon.col = mon.col)
    (end_time <- Sys.time())
    (result$gelman.R.max)
  }
  save(file=paste("fdp.jags.samples.", N.burnin, Model.File.Ext, ".Rdata", sep=""), list="result")
  save(file=paste("fdp.jags.model.", N.burnin, Model.File.Ext, ".Rdata", sep=""), list="m")
} else {
  N.burnin=2500/2 * (2**N.Retry.Loop)
  load(paste("fdp.jags.samples.", N.burnin, Model.File.Ext, ".Rdata", sep=""))
  load(paste("fdp.jags.model.", N.burnin, Model.File.Ext, ".Rdata", sep=""))

  gelman.diag(result$coda.sam, autoburnin=FALSE, multivariate = FALSE)
}
```

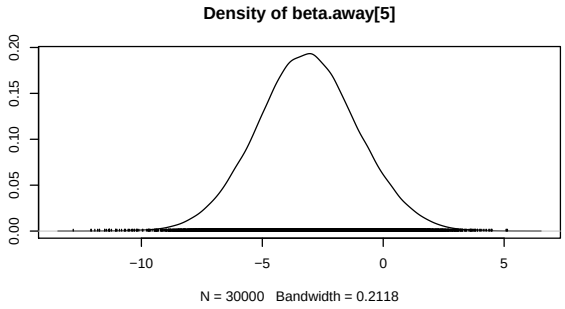
```
## Potential scale reduction factors:
##
##                  Point est. Upper C.I.
## beta.away[1]              1       1.00
## beta.away[2]              1       1.00
## beta.away[3]              1       1.00
## beta.away[4]              1       1.00
## beta.away[5]              1       1.01
## beta.away[6]              1       1.00
## beta.away[7]              1       1.00
## beta.away[8]              1       1.01
## beta.defense[1]           1       1.00
## beta.defense[2]           1       1.01
## beta.home[1]              1       1.00
## beta.home[2]              1       1.00
## beta.home[3]              1       1.00
## beta.home[4]              1       1.00
## beta.home[5]              1       1.01
## beta.home[6]              1       1.01
## beta.home[7]              1       1.01
## beta.home[8]              1       1.01
## delta[1]                  1       1.00
## delta[2]                  1       1.00
## eta[1]                    1       1.00
## eta[2]                    1       1.00
## eta[3]                    1       1.00
## eta[4]                    1       1.00
## rho[1]                    1       1.00
## rho[2]                    1       1.00
```

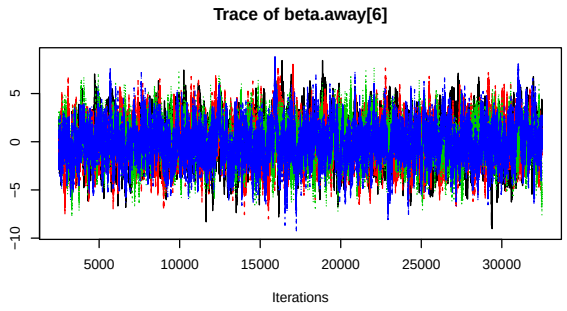
```
## rho[3]                    1       1.00
## rho[4]                    1       1.00
## sigmasq                   1       1.00
```

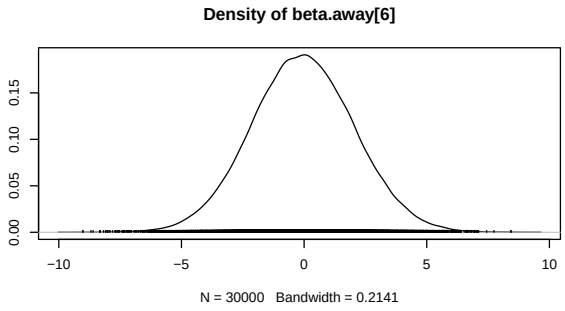
```
plot(result$coda.sam, smooth=FALSE)
```

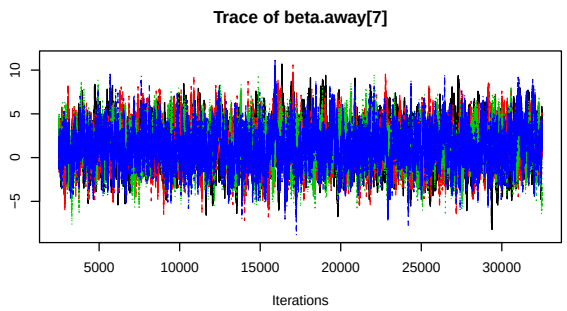


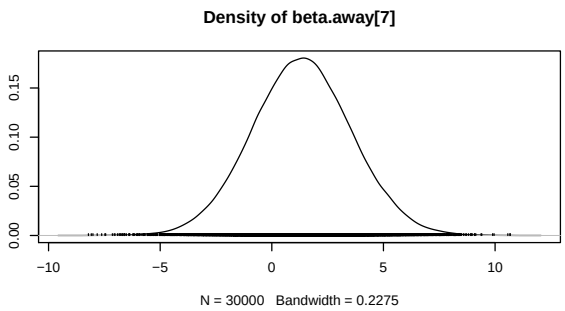

Trace of beta.away[5]
Iterations

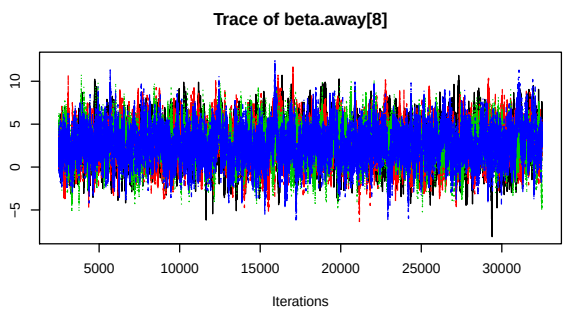

Density of beta.away[5]
N = 30000 Bandwidth = 0.2118

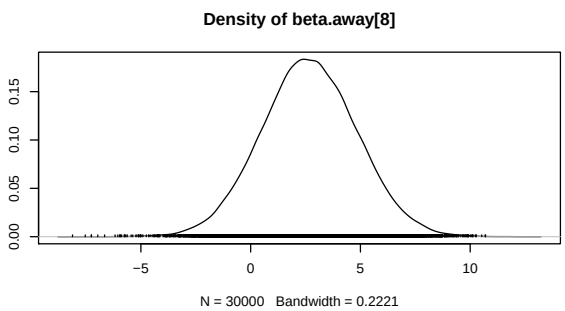

Trace of beta.away[6]
Iterations

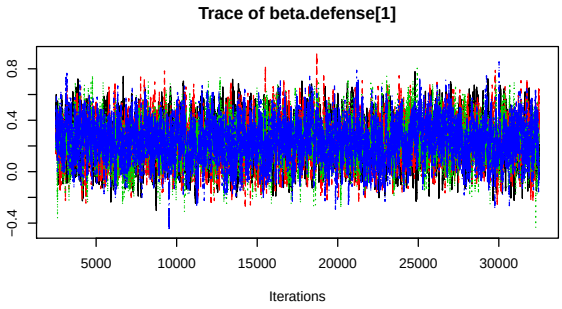

Density of beta.away[6]
N = 30000 Bandwidth = 0.2141

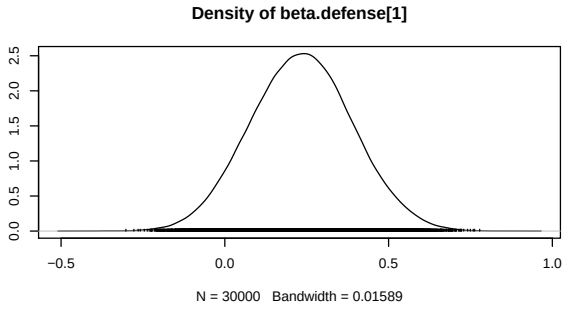

Trace of beta.away[7]
Iterations

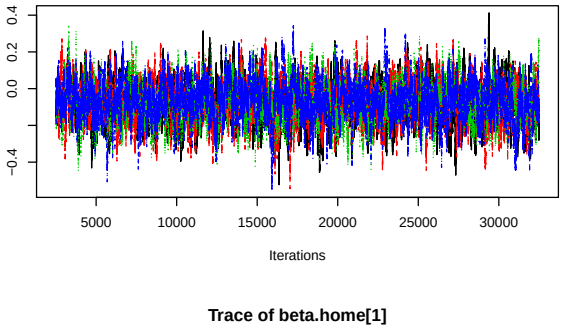

Density of beta.away[7]
N = 30000 Bandwidth = 0.2275

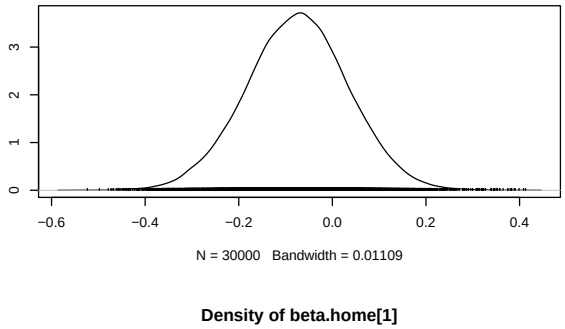

Trace of beta.away[8]
Iterations

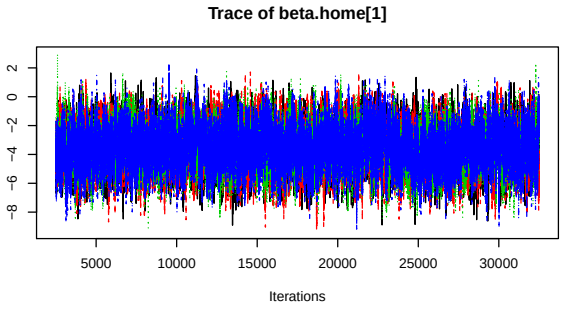

Density of beta.away[8]
N = 30000 Bandwidth = 0.2221

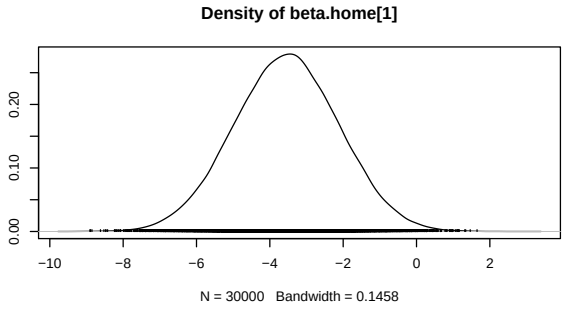

Trace of beta.defense[1]
Iterations

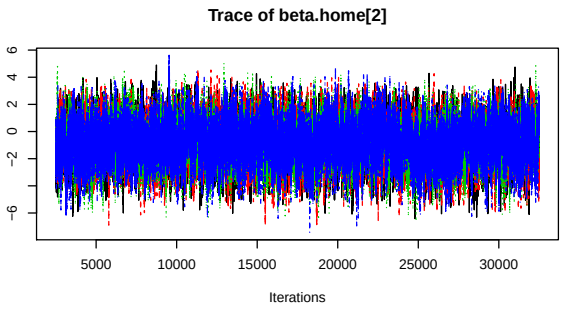

Density of beta.defense[1]
N = 30000 Bandwidth = 0.01589

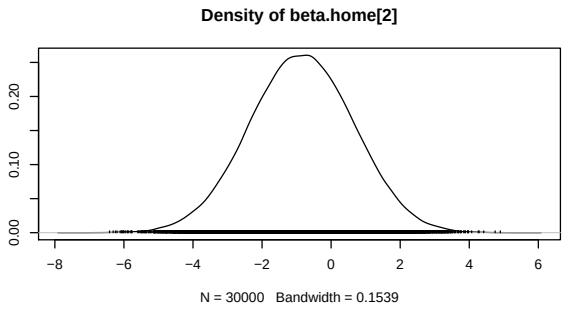

Trace of beta.defense[2]
Iterations

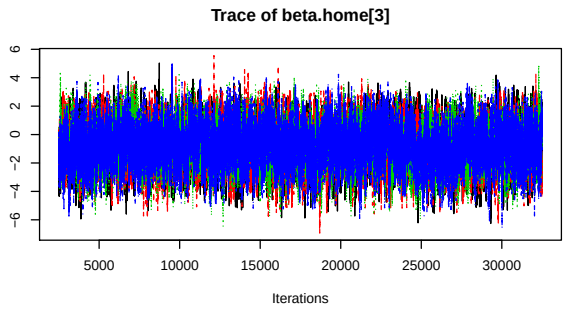

Density of beta.defense[2]
N = 30000 Bandwidth = 0.01109

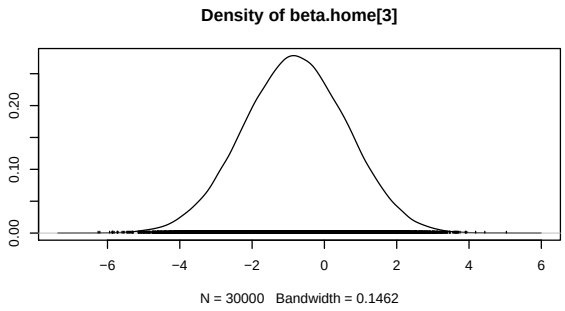

Trace of beta.home[1]
Iterations

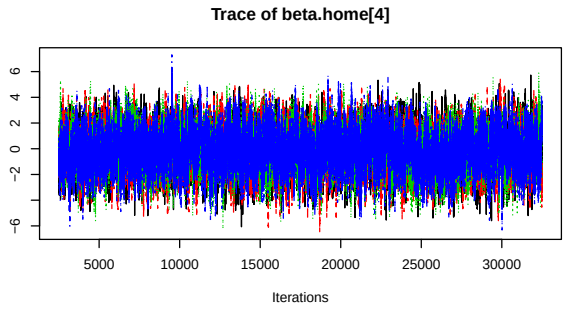

Density of beta.home[1]
N = 30000 Bandwidth = 0.1458

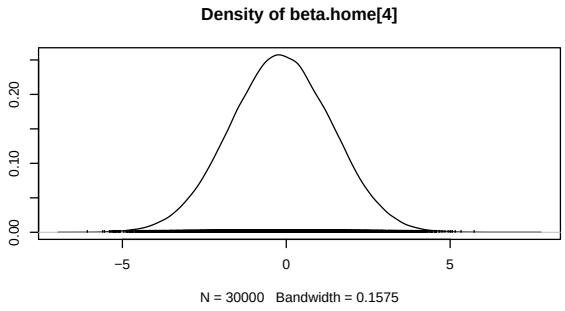

Trace of beta.home[2]
Iterations

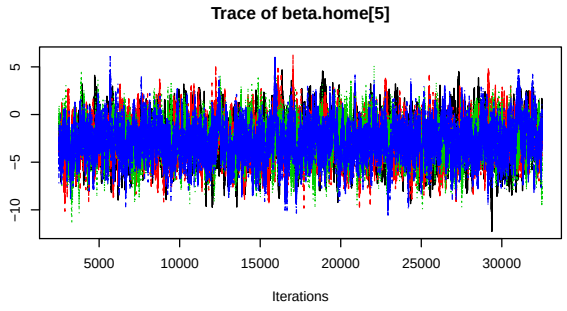

Density of beta.home[2]
N = 30000 Bandwidth = 0.1539

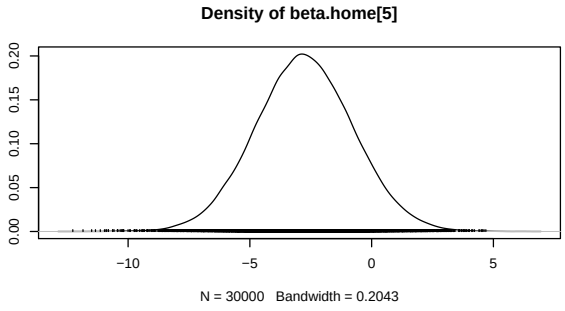

Trace of beta.home[3]
Iterations

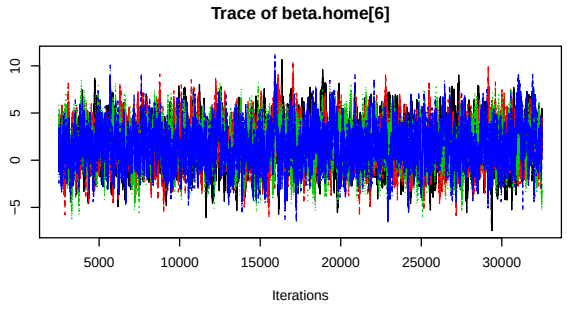

Density of beta.home[3]
N = 30000 Bandwidth = 0.1462

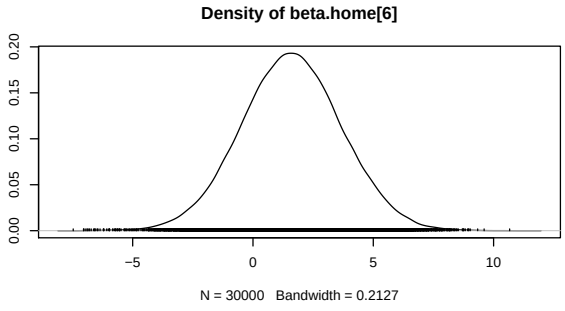

Trace of beta.home[4]
Iterations

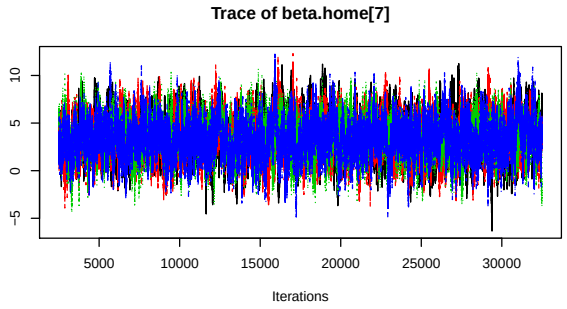

Density of beta.home[4]
N = 30000 Bandwidth = 0.1575

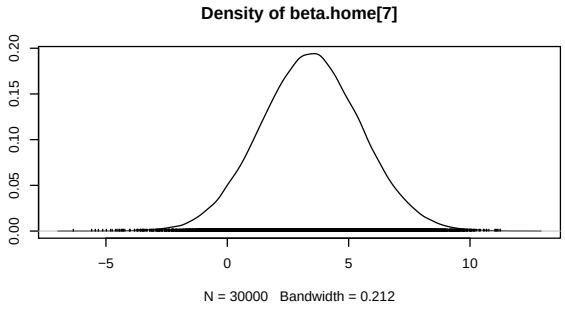

Trace of beta.home[5]
Iterations

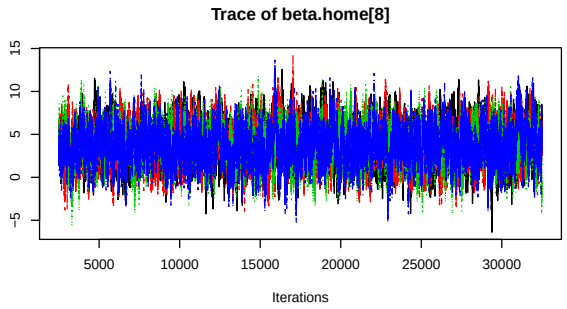

Density of beta.home[5]
N = 30000 Bandwidth = 0.2043

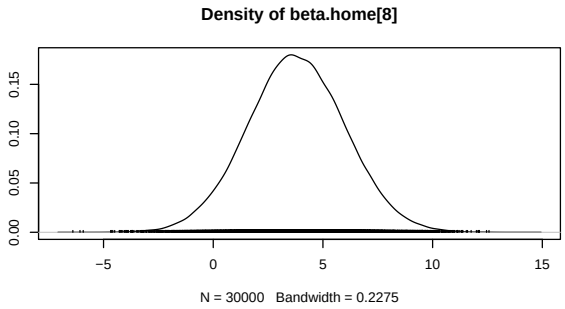

Trace of beta.home[6]
Iterations

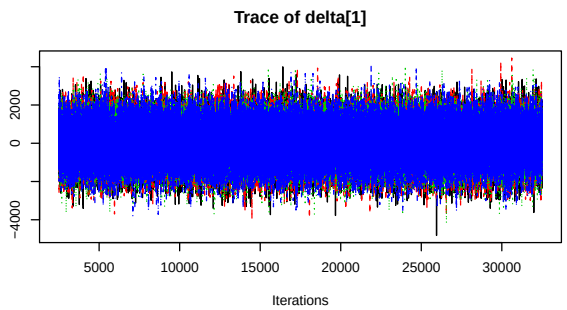

Density of beta.home[6]
N = 30000 Bandwidth = 0.2127

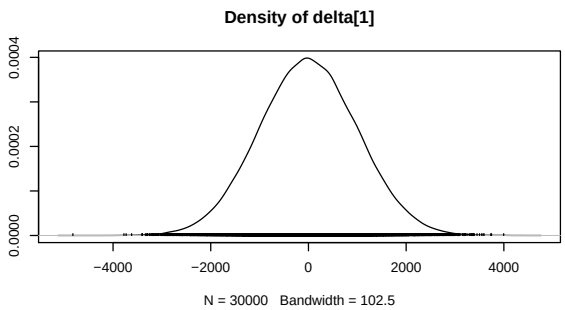

Trace of beta.home[7]
Iterations

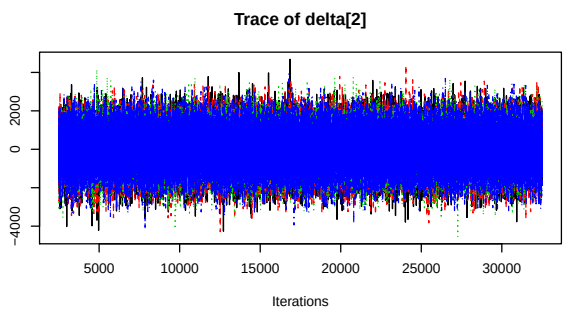

Density of beta.home[7]
N = 30000 Bandwidth = 0.212

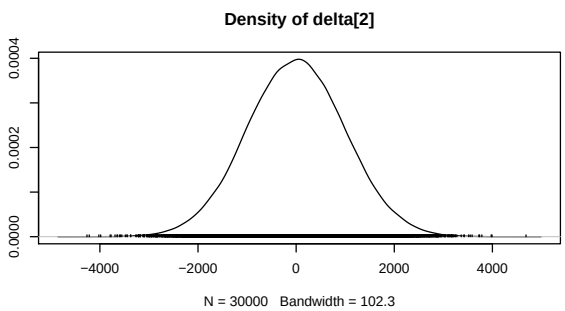

Trace of beta.home[8]
Iterations

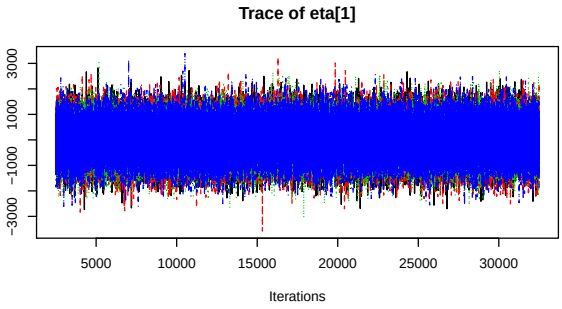

Density of beta.home[8]
N = 30000 Bandwidth = 0.2275

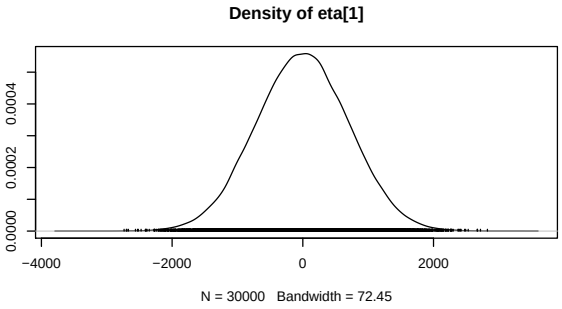

Trace of delta[1]
Iterations

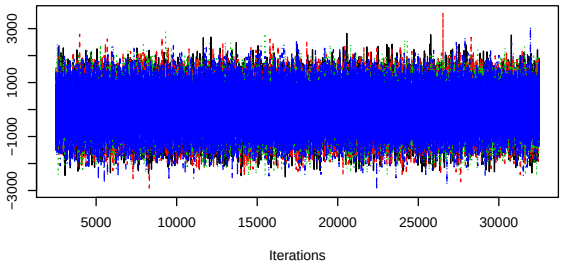

Density of delta[1]
N = 30000 Bandwidth = 102.5

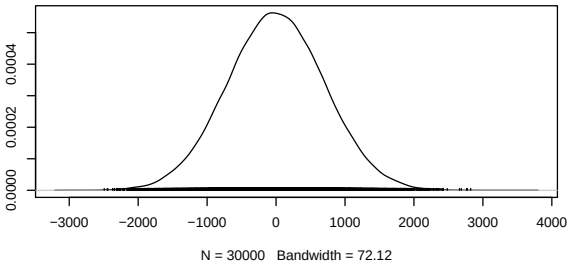

Trace of delta[2]
Iterations

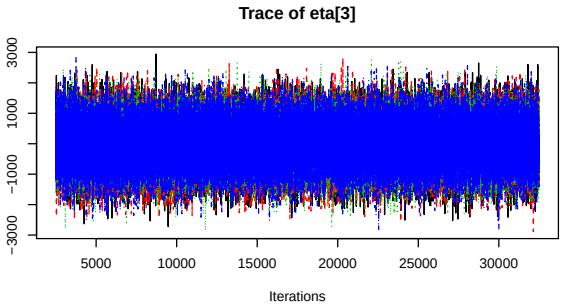

Density of delta[2]
N = 30000 Bandwidth = 102.3

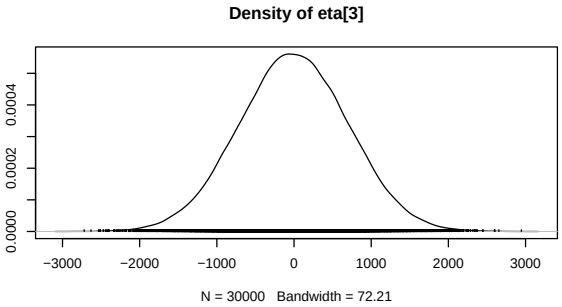

Trace of eta[1]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

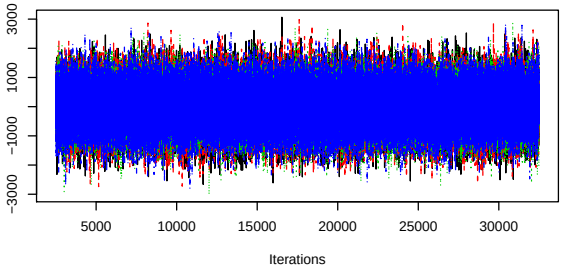

Density of eta[1]
0.0004
0.0002
0.0000
−4000
−2000
0
2000
N = 30000 Bandwidth = 72.45

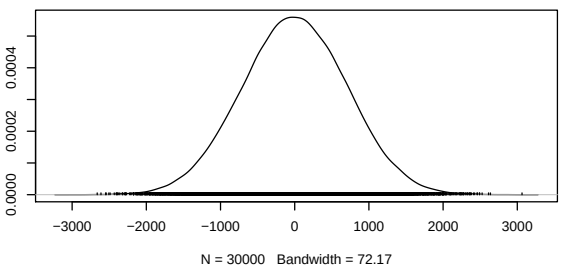

Trace of eta[2]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

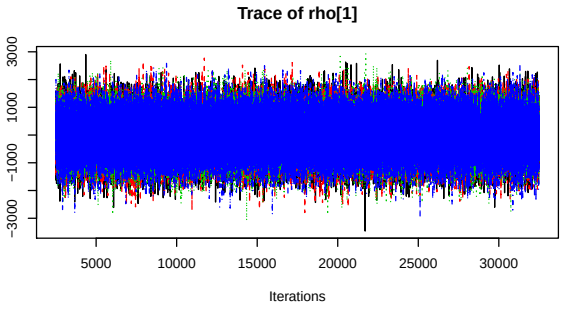

Density of eta[2]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
4000
N = 30000 Bandwidth = 72.12

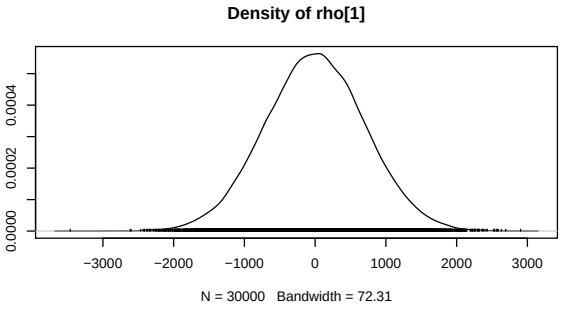

Trace of eta[3]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

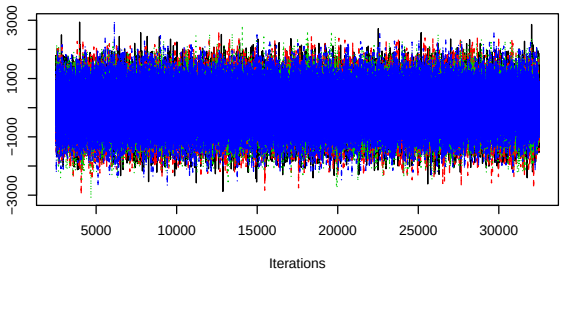

Density of eta[3]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.21

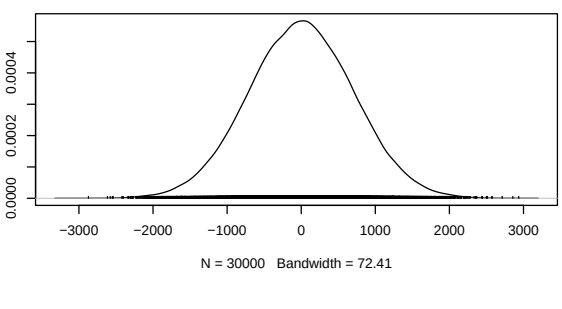

Trace of eta[4]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

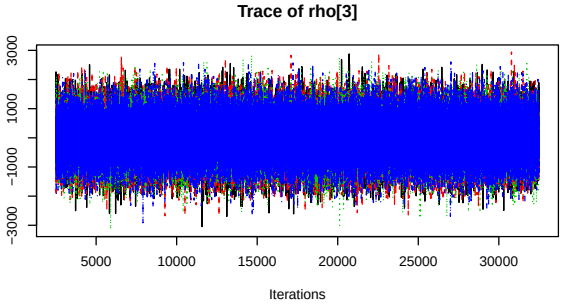

Density of eta[4]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.17

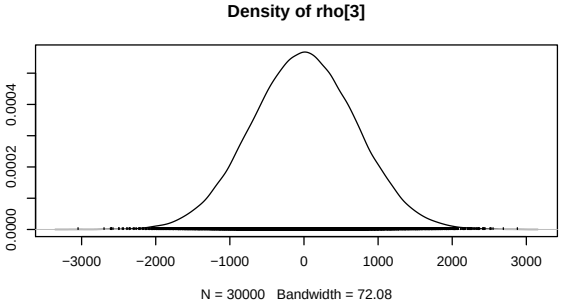

Trace of rho[1]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

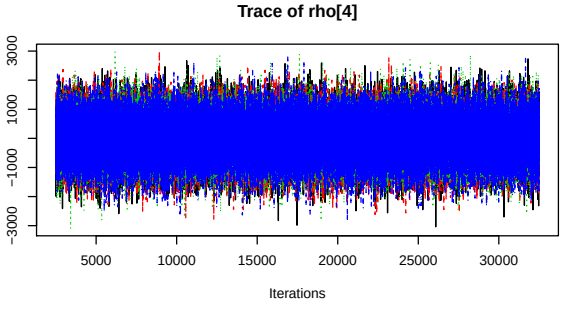

Density of rho[1]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.31

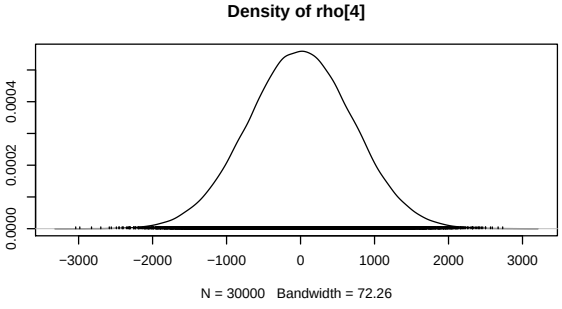

Trace of rho[2]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

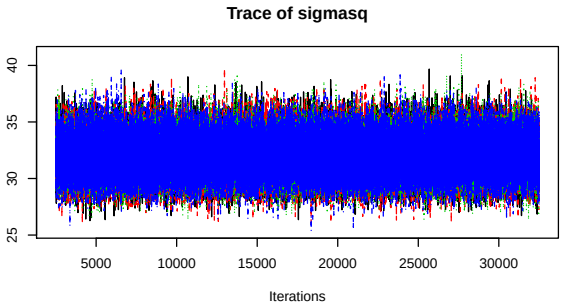

Density of rho[2]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.41

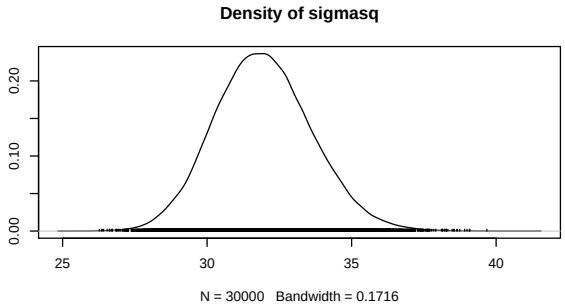

Trace of rho[3]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations


Density of rho[3]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.08


Trace of rho[4]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations


Density of rho[4]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.26





Converged as `gelman.R.max` $= 1.002 < 1.1$ and the plot also looks good.

```
summary(result$coda.sam)
```

```
##
## Iterations = 2501:32500
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 30000
##
```

```
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                     Mean       SD Naive SE Time-series SE
## beta.away[1]    -4.4636    1.474 0.004256        0.02372
## beta.away[2]    -2.4624    1.458 0.004210        0.02317
## beta.away[3]    -0.8332    1.524 0.004399        0.02341
## beta.away[4]    -0.9295    1.506 0.004348        0.02333
## beta.away[5]    -3.1336    2.092 0.006039        0.05109
## beta.away[6]    -0.0485    2.095 0.006047        0.04977
## beta.away[7]     1.3464    2.235 0.006451        0.05245
## beta.away[8]     2.6680    2.180 0.006292        0.05058
## beta.defense[1]  0.2348    0.155 0.000449        0.00285
## beta.defense[2] -0.0750    0.110 0.000316        0.00286
## beta.home[1]    -3.5465    1.427 0.004119        0.02297
## beta.home[2]    -0.8547    1.505 0.004346        0.02281
## beta.home[3]    -0.8048    1.430 0.004129        0.02218
## beta.home[4]    -0.1613    1.541 0.004447        0.02484
## beta.home[5]    -2.7594    2.013 0.005810        0.04794
## beta.home[6]     1.6027    2.088 0.006028        0.05020
## beta.home[7]     3.4434    2.079 0.006002        0.04957
## beta.home[8]     3.7885    2.228 0.006430        0.05187
## delta[1]         0.5124 1002.753 2.894698        2.90765
## delta[2]        -0.5926 1000.900 2.889350        2.90995
## eta[1]          -4.4295  708.899 2.046415        2.04643
## eta[2]          -0.1967  705.636 2.036996        2.08127
## eta[3]          -2.6215  706.540 2.039605        2.05043
## eta[4]          -0.0937  706.160 2.038509        2.03851
## rho[1]          -2.1788  707.476 2.042306        2.05159
## rho[2]           0.1973  708.476 2.045195        2.03842
## rho[3]           1.1455  705.234 2.035836        2.02869
## rho[4]           0.0402  706.971 2.040849        2.03517
## sigmasq         31.9444    1.679 0.004846        0.00504
##
## 2. Quantiles for each variable:
##
##                      2.5%      25%     50%       75%    97.5%
## beta.away[1]       -7.3597   -5.458 -4.4632  -3.46216   -1.597
## beta.away[2]       -5.3210   -3.447 -2.4598  -1.47056    0.381
## beta.away[3]       -3.8368   -1.859 -0.8279   0.19686    2.137
## beta.away[4]       -3.8975   -1.949 -0.9238   0.09688    1.993
## beta.away[5]       -7.2200   -4.526 -3.1435  -1.74877    1.016
## beta.away[6]       -4.1384   -1.459 -0.0593   1.35282    4.092
## beta.away[7]       -3.0286   -0.151  1.3423   2.83241    5.763
## beta.away[8]       -1.5715    1.214  2.6541   4.12679    6.974
## beta.defense[1]    -0.0662    0.129  0.2348   0.33967    0.541
## beta.defense[2]    -0.2929   -0.147 -0.0740  -0.00212    0.139
## beta.home[1]       -6.3444   -4.514 -3.5429  -2.58185   -0.756
## beta.home[2]       -3.8255   -1.873 -0.8525   0.16951    2.075
## beta.home[3]       -3.6191   -1.773 -0.8028   0.16886    1.979
## beta.home[4]       -3.1849   -1.202 -0.1595   0.88608    2.834
## beta.home[5]       -6.6870   -4.100 -2.7698  -1.42217    1.224
## beta.home[6]       -2.4973    0.204  1.5951   2.99237    5.717
## beta.home[7]       -0.6236    2.051  3.4406   4.83075    7.546
```

```
## beta.home[8]       -0.5856    2.302  3.7748   5.28452    8.170
## delta[1]        -1957.5020 -678.601 -2.3096 675.59568 1974.721
## delta[2]        -1962.0265 -677.623  0.3306 672.01082 1973.059
## eta[1]          -1393.2636 -484.157 -3.8283 475.09084 1380.327
## eta[2]          -1383.1520 -475.871 -1.3895 475.35275 1388.076
## eta[3]          -1390.5465 -476.938 -4.0451 475.59379 1379.797
## eta[4]          -1380.4350 -476.480 -0.7876 478.51307 1384.202
## rho[1]          -1393.2531 -477.516 -1.6231 474.21504 1387.126
## rho[2]          -1388.6359 -478.087  1.1899 478.14215 1389.690
## rho[3]          -1381.9251 -474.961  0.5987 476.30937 1384.538
## rho[4]          -1391.9517 -474.983  0.8282 476.87848 1383.491
## sigmasq            28.8029   30.787 31.8924  33.03684   35.382
```

Effective Sample Size

```
(eff.size = effectiveSize(result$coda.sam[, ]))
```

```
##    beta.away[1]    beta.away[2]    beta.away[3]    beta.away[4]    beta.away[5]
##            3879            3980            4257            4184            1681
##    beta.away[6]    beta.away[7]    beta.away[8] beta.defense[1] beta.defense[2]
##            1775            1821            1863            2987            1469
##    beta.home[1]    beta.home[2]    beta.home[3]    beta.home[4]    beta.home[5]
##            3888            4381            4176            3859            1765
##    beta.home[6]    beta.home[7]    beta.home[8]        delta[1]        delta[2]
##            1733            1766            1847          118955          118320
##          eta[1]          eta[2]          eta[3]          eta[4]          rho[1]
##          120000          115504          118747          120000          118984
##          rho[2]          rho[3]          rho[4]         sigmasq
##          120813          120869          120681          111066
```

The effective sample sizes of all parameters are greater than 400.

Probabiliy of players perform better at home than away

```
post.samp = as.matrix(result$coda.sam)

beta.home = post.samp[, paste("beta.home[",1:(Num.Rank*Num.Position),"]", sep="")]
beta.away = post.samp[, paste("beta.away[",1:(Num.Rank*Num.Position),"]", sep="")]

prob.home.away = rep(0, Num.Rank * Num.Position)
for (r in 1:Num.Rank) {
  for (p in 1:Num.Position) {
    idx = (p-1) * Num.Rank + r
    prob.home.away[idx] = mean(beta.home[, idx] > beta.away[, idx])
  }
}
prob.home.away
```

```
## [1] 0.8082 0.9165 0.5100 0.7491 0.6302 0.9239 0.9498 0.7928
```

```
prob.home.away.df = data.frame(colnames(X.home))
prob.home.away.df$prob.home.bt.away = prob.home.away
```

```
colnames(prob.home.away.df) = c("Rank:Position", "Prob.home.bt.away")

kable(prob.home.away.df)
```

Rank:Position	Prob.home.bt.away
RankRank1:PositionPK	0.8082
RankRank2:PositionPK	0.9165
RankRank3:PositionPK	0.5100
RankRank4:PositionPK	0.7491
RankRank1:PositionQB	0.6302
RankRank2:PositionQB	0.9239
RankRank3:PositionQB	0.9498
RankRank4:PositionQB	0.7928

Beta defense

If a player is facing a team which gives up more points to players on average, we expect the player will score more points.

```
beta.defense = post.samp[, paste("beta.defense[",1:Num.Position,"]", sep="")]

beta.defense.int.df = data.frame(colnames(X.defense))
beta.defense.int = matrix(rep(0, Num.Position * 4), nrow=Num.Position, ncol = 4)

int.alpha=0.05
for (p in 1:Num.Position) {
  beta.defense.int[p, 1:3] =  quantile(beta.defense[, p], c(int.alpha/2, 0.5, 1-int.alpha/2))
  beta.defense.int[p, 4] = mean(beta.defense[, p])
}

beta.defense.int.df$pct025 = beta.defense.int[, 1]
beta.defense.int.df$pct975 = beta.defense.int[, 3]
beta.defense.int.df$median = beta.defense.int[, 2]
beta.defense.int.df$mean = beta.defense.int[, 4]
colnames(beta.defense.int.df) = c("beta.defense.position", "pct025", "pct975", "median", "mean")
kable(beta.defense.int.df)
```

beta.defense.position	pct025	pct975	median	mean
AvgOppPAP7Wks:PositionPK	-0.0662	0.5415	0.2348	0.2348
AvgOppPAP7Wks:PositionQB	-0.2929	0.1386	-0.0740	-0.0750

We observe that the median beta.defense for PK is positive as expected. But for QB, it is negative, that implies QB actually score less against bad defensive team.

DIC

```
(dic.samp = dic.samples(m, NSim))
```

```
## Mean deviance:  4700
```

```
## penalty 19
## Penalized deviance: 4719
```

Alternative Model - no rank

```
#sink("fdp.norank.bug")
#cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                 + inprod(X.home[i, ], beta.home)
                 + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # The entry of the beta.defense corresponds to Opponent:Position
  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    beta.defense[p] ~ dnorm(delta[p], 1/1000^2)
    delta[p] ~ dnorm(0, 1/100000^2)
  }

  # The entry of the beta.home and beta.away corresponds to Position
  # In our model, we pool the beta.home/away based on Position
  for (t in 1:Num.Position) {
    beta.home[t] ~ dnorm(eta, 1/1000^2)
    beta.away[t] ~ dnorm(rho, 1/1000^2)
  }
  eta ~ dnorm(0, 1/100000^2)
  rho ~ dnorm(0, 1/100000^2)

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}
#    ",fill = TRUE)
#sink()
```